ภาษาซีกับแอสเซมบลี

■ จเร เลิศสุดวิชัย

เนื่องจากมุมภาษาซีในตอนที่แล้วซึ่งเป็นตอนที่หนึ่งของเทคนิคการเขียนโปรแกรมเรสซิเดนต์ ในโปรแกรมมีการเชื่อมโยงกันระหว่างภาษาซีกับภาษาแอสเซมบลี ปรากฏว่ามีผู้อ่านสนใจถามถึงวิธีการเขียนโปรแกรมภาษาซีเชื่อมโยงกับภาษาแอสเซมบลีกันมา ดังนั้นในมุมภาษาซีฉบับนี้ซึ่งตามกำหนดจะต้องเป็นตอนที่ 2 ของเทคนิคการเขียนโปรแกรมเรสซิเดนต์ขอเลื่อนตอนที่สองเอาไว้เป็นฉบับต่อไป และฉบับนี้จะเป็นวิธีการเขียนโปรแกรมภาษาซีเชื่อมโยงกับภาษาแอสเซมบลีแทนครับ ยังไงก็ต้องขอให้ท่านผู้อ่านที่ติดตามการเขียนโปรแกรมเรสซิเดนต์อดใจรอเอาไว้ก่อน

ทำไมต้องภาษาแอสเซมบลี

มีเหตุผลด้วยกันหลายอย่างในการที่เขียนโปรแกรมภาษาซีแล้วเรียกโปรแกรมภาษาแอสเซมบลี เช่น เพื่อเพิ่มความเร็วให้กับโปรแกรม ถึงแม้ปัจจุบันคอมพายเลอร์ภาษาซีหลายตัว มีความสามารถในการคอมพายล์โปรแกรมแล้วได้ภาษาเครื่องที่สั้นซึ่งทำให้ได้โปรแกรมภาษาซีที่ทำงานได้รวดเร็ว แต่อย่างไรก็ตามไม่มีคอมพายเลอร์ตัวใดมีความสามารถแปลโปรแกรมภาษาซีให้ได้ภาษาเครื่องที่สั้นเทียบเท่ากับนักเขียนโปรแกรมผู้ชำนาญเขียนโปรแกรมด้วยภาษาแอสเซมบลี ฮาร์ดแวร์หรือการควบคุมอุปกรณ์บางอย่างต้องการเวลาที่แน่นอน ดังนั้น จำเป็นที่โปรแกรมส่วนนี้ต้องเขียนด้วยภาษาแอสเซมบลีเพื่อกำหนดเวลาให้ได้ตามที่ต้องการ หรือโปรแกรมในบางส่วนซึ่งอาจจะต้องมีการเปลี่ยนรีจิสเตอร์เซกเมนต์ (Segment Register) บางตัว เช่นเปลี่ยนค่าในดาต้าเซกเมนต์เพื่ออ้างอิงถึงข้อมูลส่วนอื่น ในกรณีไม่สามารถทำด้วยภาษาซีได้ และอีกหลาย ๆ สาเหตุ ในกรณีเพื่อเพิ่มความเร็วของโปรแกรมภาษาซี โปรแกรมส่วนใดที่ต้องเขียนด้วยแอสเซมบลี ก็คือในส่วนที่ถูกเรียกใช้งานบ่อยครั้งมากและมีผลกระทบต่อเวลาของการทำงานของโปรแกรม ตัวอย่างเช่น การคำนวณทางคณิตศาสตร์พวก Floating-point, กราฟิก (Graphic) หรือรูปภาพ เป็นต้น

ในบทความนี้เป็นการชี้แนะถึงวิธีการเขียนโปรแกรมภาษาซีแล้วเรียกใช้โปรแกรมภาษาแอสเซมบลี ตัวอย่างโปรแกรมก็จะมีทั้งเทอร์โบซีและไมโครซอฟต์ซี เครื่องมือที่ต้องใช้มีไมโครซอฟต์มาโครแอสเซมเบลอร์ (MASM) เวอร์ชัน 3 ขึ้นไป หรือเทอร์โบซีแอสเซมเบลอร์ และผู้จะทดลองโปรแกรมตัวอย่างควรจะเป็นผู้มีความคุ้นเคยภาษาแอสเซมบลีพอสมควร

สิ่งที่ควรจะรู้เพื่อใช้ในการเขียนโปรแกรมภาษาซีเชื่อมโปรแกรมภาษาแอสเซมบลี ซึ่งจะกล่าวเป็นหัวข้อย่อยฉัดต่อไปนั้นก็ได้แก่

- โมเดล (Model) ของโปรแกรมภาษาซี กับการจัดหน่วยความจำหรือเซกเมนต์ของโปรแกรม
- การส่งผ่านพารามิเตอร์ของภาษาซีในการเรียกฟังก์ชัน
- การส่งผ่านค่ากลับของฟังก์ชัน

โมเดลของภาษาซี

ในการเขียนโปรแกรมคอมพิวเตอร์นั้นสามารถจะเชื่อมโยงหรือลิงค์ระหว่างโปรแกรมของแต่ละภาษาได้กันหมด สิ่งที่จะต้องรู้เพื่อใช้ในการเชื่อมโยงคือการจัดหน่วยความจำของโปรแกรม เช่นเดียวกับในการเขียนโปรแกรมภาษาแอสเซมบลีให้ภาษาซีเรียก จึงจำเป็นต้องรู้ถึงการจัดจำหน่วยความจำของโปรแกรมภาษาซี เพื่อที่จะจัดลักษณะการใช้หน่วยความจำของภาษาแอสเซมบลีให้ตรงกัน

โปรแกรมในภาษาซี การจัดเรียงหน่วยความจำของโปรแกรมมีหลายรูปแบบ แบ่งออกได้ 6 โมเดล คือ Tiny, small, medium, compact, large และ huge สำหรับเทอร์โบซีนั้นมีครบทั้งหมดทุกโมเดล ส่วนไมโครซอฟต์ซีมีเพียง 5 โมเดล โมเดลที่ขาดไปคือ Tiny ซึ่งต่อไปเราจะมาพิจารณากันถึงรายละเอียดและโมเดลกันก่อน

- แบบ Tiny เป็นแบบที่เล็กที่สุดของการจัดหน่วยความจำ ทั้งโปรแกรม ข้อมูล และสแต็กอยู่ภายในเซกเมนต์เดียวกัน คือ ไม่เกิน 64 กิโลไบต์ (CS, DS, SS และ ES ชี้ที่เดียวกัน)

ซึ่งโปรแกรมในแบบ Tiny สามารถจะแปลงให้เป็นฟอร์แมต .COM ได้

- แบบ Small โปรแกรมในแบบนี้นั้นโปรแกรมและข้อมูลแยกออกจากกันคนละเซกเมนต์ ซึ่งขนาดของโปรแกรมมีได้เป็น 64 กิโลไบต์ แต่ส่วนของข้อมูล (DS) สแต็ก (SS) และเอกซ์ต้าเซกเมนต์ (ES) ชี้ที่เซกเมนต์เดียวกัน ทำให้ทั้งข้อมูลและสแต็กมีขนาดได้ไม่เกิน 64 กิโลไบต์

- แบบ Medium โปรแกรมในแบบนี้นั้นข้อมูลจะถูกกำหนดให้อยู่ในเซกเมนต์เดียว แยกจากโปรแกรมซึ่งจะมีขนาดได้ 64 กิโลไบต์ แต่โปรแกรมมีได้หลายเซกเมนต์ ขนาดของโปรแกรมมีได้จนถึง 1 เมกะไบต์ ซึ่งโปรแกรมแบบ Medium เหมาะสมกับโปรแกรมที่มีขนาดใหญ่ แต่ใช้ข้อมูลน้อย

- แบบ Compact โปรแกรมแบบ Compact การจัดการหน่วยความจำจะตรงข้ามกับแบบ Medium คือโปรแกรมจะถูกจำกัดอยู่เพียงเซกเมนต์เดียว แต่ข้อมูลมีได้

หลายเซกเมนต์

- แบบ Large โปรแกรมแบบนี้ ทั้งข้อมูลและโปรแกรมมีได้หลายเซกเมนต์ ซึ่งเหมาะสำหรับโปรแกรมขนาดใหญ่

- แบบ Huge โปรแกรมลักษณะนี้คล้ายกับโปรแกรมแบบ Large ซึ่งในโปรแกรมแบบ Huge อนุญาตให้ใช้ข้อมูลแบบสแตติกได้เกิน 64 กิโลไบต์ได้

ในการเขียนภาษาแอสเซมบลีก็จะต้องจัดเรียงหน่วยความจำให้ถูกต้องและตรงกันกับโมเดลของภาษาซี สำหรับการจัดเซกเมนต์ของโปรแกรมตามโมเดล เซกเมนต์ของโปรแกรมและข้อมูลจะมีชื่อที่เป็นมาตรฐานของไมโครซอฟต์ เช่นในตารางที่ 1 เป็นการตั้งชื่อเซกเมนต์ของเทอร์โบซี ส่วนตารางที่ 2 เป็นของไมโครซอฟต์ซี

เอาละครับมาถึงตอนนี้ก็พอจะเขียนโปรแกรมภาษาแอสเซมบลีให้ภาษาซีเรียกอย่างง่าย ๆ ได้แล้วละครับ ซึ่งพอจะแบ่งวิธีการเขียนได้ 2 ลักษณะ คือ อาศัยรูปแบบหรือ Template ภาษาแอสเซมบลีให้ตรงกับการกำหนดเซกเมนต์ของโปรแกรมภาษาซีตามโมเดลนั้น ๆ และการหา Template อย่างง่ายที่สุดก็ทำได้โดยการเขียนโปรแกรมภาษาซีสั้น ๆ แล้วคอมพายล์ให้ได้ไฟล์แอสเซมบลี (ใส่-s ใน option การคอมพายล์สำหรับเทอร์โบซีและใส่ -Fa สำหรับไมโครซอฟต์ซี) ตัดเอาส่วนไม่จำเป็นออก ซึ่ง Template ที่จากเทอร์โบซีใน Tiny โมเดลอยู่ในรูปที่ 1 และ Template สำหรับไมโครซอฟต์ซีใน small โมเดลอยู่ในรูปที่ 2 นำ Template ที่ได้มาเป็นต้นแบบการเขียนโปรแกรมภาษาแอสเซมบลีได้ต่อไป ลองเขียนโปรแกรมง่าย ๆ ก่อนดังในโปรแกรมที่ 1 รูปที่ 3 โปรแกรม CCC1.C เป็นโปรแกรมสั้น ๆ ที่เรียกไปยังโปรแกรมแอสเซมบลีโดยก่อนและหลังเรียกจะพิมพ์ข้อความบอกก่อนเข้าฟังก์ชันที่เป็นแอสเซมบลี ส่วนโปรแกรมเอสแซลบลีนั้นเป็นโปรแกรมสำหรับไมโครซอฟต์ซี (สำหรับเทอร์โบซีก็ทำคล้ายกัน จะไม่ขอกล่าวถึง) รูปที่ 4 โปรแกรม ASM11.C ภายในฟังก์ชันก็จะพิมพ์ข้อความว่าเข้ามาอยู่ในฟังก์ชันของแอสเซมบลีแล้ว การคอมพายล์และผลลัพธ์อยู่ในรูปที่ 5 วิธีการการเขียนโปรแกรมแอสเซมบลีเชื่อมโยงกับภาษาซีโดยอาศัย Template ตามที่กล่าวมาแล้วนั้นเป็นวิธีการพื้นฐานและเป็นวิธีการแรกเริ่มอันหนึ่ง แต่ในปัจจุบันแอสเซมเบลอร์ที่ออกมาใหม่ได้เพิ่มขีดความสามารถขึ้น เช่น มาโครแอสเซมเบลอร์ เวอร์ชัน 5.0 (MASM 5.0) หรือ เทอร์โบแอสเซมเบลอร์ (TASM) ซึ่งจะช่วยให้การเขียนโปรแกรมแอสเซมบลีสะดวกและง่ายขึ้น

ในการเขียนโปรแกรมแอสเซมบลีในลักษณะที่ 2 จะอาศัยความสามารถของแอสเซมบลีในการจัดเซกเมนต์หรือความจำให้โดยการให้ไดเรคทีฟ (Directive) เป็นตัวกำหนด ไดเรคทีฟที่เราจะใช้มีดังนี้

- การกำหนดให้จัดเรียงเซกเมนต์

ตารางที่ 1 ชื่อเซกเมนต์ของโปรแกรมที่คอมพายล์ด้วยเทอร์โบซี

โมเดล	โปรแกรม (code)	ข้อมูล (Data)
Tiny, small	_TEXT	_DATA
Medium	filename_TEXT	_DATA
Compact	_TEXT	filename_DATA
Large	filename_TEXT	_DATA
Huge	filename_TEXT	filename_DATA

ตารางที่ 2 ชื่อเซกเมนต์ของโปรแกรมที่คอมพายล์ด้วยไมโครซอฟต์ซี

โมเดล	โปรแกรม (code)	ข้อมูล (Data)
Small	_TEXT	_DATA
Medium	filename_TEXT	_DATA
Compact	_TEXT	_DATA*
Large	filename_TEXT	_DATA*
Huge	filename_TEXT	_DATA*

หมายเหตุ * ชื่อเซกเมนต์ default คือ _DATA ส่วนชื่อเซกเมนต์อื่นแล้วแต่กำหนดขึ้นของแต่ละไฟล์

รูปที่ 1 Template สำหรับเขียนแอสเซมบลีซึ่งได้เทอร์โบซี โมเดล Tiny

```
;---------Start of Segment defination -----------

_TEXT   segment byte public 'CODE'
DGROUP  group   _DATA,_BSS
        assume  cs:_TEXT,ds:DGROUP,ss:DGROUP
_TEXT   ends
_DATA   segment word public 'DATA'
_DATA   ends
_BSS    segment word public 'BSS'
_BSS    ends

;---------End of Segment defination -------------

_DATA   segment

;---------Put your initialized data in here ------

_DATA   ends

_BSS    segment

;---------Put your uninitialized data in here --------

_BSS    ends


_TEXT   segment

;---------Put your code in here -------------------

_TEXT   ends


        end
```

รูปที่ 2 Template สำหรับเขียนแอสเซมบลีซึ่งได้จาก ไมโครซอฟต์ซี โมเดล small

```
;----------Start of Segment defination -----------

_TEXT   SEGMENT  WORD PUBLIC 'CODE'
_TEXT   ENDS
_DATA   SEGMENT  WORD PUBLIC 'DATA'
_DATA   ENDS
CONST   SEGMENT  WORD PUBLIC 'CONST'
CONST   ENDS
_BSS    SEGMENT  WORD PUBLIC 'BSS'
_BSS    ENDS
DGROUP  GROUP   CONST, _BSS, _DATA
        ASSUME  CS: _TEXT, DS: DGROUP, SS: DGROUP

;---------- End of Segment defination ------------

_DATA   SEGMENT

;---------Put your initialized data in here ---------

_DATA   ENDS

CONST   SEGMENT

;------------Put your constant in here ------------

CONST   ENDS

_BSS    SEGMENT

;--------Put your uninitialized data in here -------
```

รูปที่ 2 (ต่อ)

```
_BSS    ENDS


_TEXT       SEGMENT
        ASSUME  CS: _TEXT
;----------Put your code in here -----------------

_TEXT   ENDS

END
```

รูปที่ 3 CCC1.C โปรแกรมภาษาซีตัวอย่างที่ 1

```
/****** First Example CCC1.C ******/
#include <stdio.h>

extern  void call_asm (void);

main ()
{
   puts ("Before call assambly function");
   call_asm ();
   puts ("Return from assambly function");
}
```

รูปที่ 4 ASM11.ASM โปรแกรมภาษาแอสเซมบลีตัวอย่างที่ 1 ซึ่งเขียนโดยใช้ Template ช่วย

```
;*************** ASM11.asm *************

;----------Start of Segment defination ------------

_TEXT   SEGMENT  WORD PUBLIC 'CODE'
_TEXT   ENDS
_DATA   SEGMENT  WORD PUBLIC 'DATA'
_DATA   ENDS
CONST   SEGMENT  WORD PUBLIC 'CONST'
CONST   ENDS
_BSS    SEGMENT  WORD PUBLIC 'BSS'
_BSS    ENDS
DGROUP  GROUP   CONST, _BSS, _DATA
        ASSUME  CS: _TEXT, DS: DGROUP, SS: DGROUP

;----------End of Segment defination -------------

_DATA   SEGMENT

string          db  'Now in assambly function',13,10,'$'

_DATA   ENDS


_TEXT       SEGMENT
        ASSUME  CS: _TEXT

        PUBLIC  _call_asm

_call_asm       proc    near
                mov     dx,offset DGROUP : string
                mov     ah,09h
                int     21h
                ret
_call_asm       endp
_TEXT   ENDS

END
```

รูปที่ 5 การคอมพายล์และผลลัพธ์ที่ได้จากโปรแกรมตัวอย่าง CCC1.C กับ ASM11.C และ CCC1.C กับ ASM12.C

```
C>masm -Ml asm11;

Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.
All rights reserved.

  51362 + 214782 Bytes symbol space free

       0 Warning Errors
       0 Severe  Errors

C>tcc -ms ccc1.c asm11.obj
Turbo C  Version 2.0  Copyright (c) 1987,
1988 Borland International
ccc1.c:
Turbo Link  Version 2.0  Copyright (c) 1987,
1988 Borland International

        Available memory 196510

C>ccc1
Before call assambly function
Now in assambly function
Return from assambly function

C>masm -Ml asm12;

Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.
All rights reserved.

  51362 + 214782 Bytes symbol space free

       0 Warning Errors
       0 Severe  Errors

C>cl -AS ccc1.c asm12

Microsoft (R) C Optimizing Compiler Version 5.00
Copyright (C) Microsoft Corp 1984,1985,,1986,1987.
All rights reserved.
ccc1.c

Microsoft (R) Overlay Linker  Version 3.61
Copyright (C) Microsoft Corp 1983-1987.
All rights reserved.
Object Modules [.OBJ]: CCC1.OBJ+ASM12
Run File [CCC1.EXE]: CCC1.EXE /NOI
List File [NUL.MAP]: NUL
Libraries [.LIB]: ;


C>ccc1
Before call assambly function
Now in assambly function
Return from assambly function
```

ตามดอสด้วยไดเรคทีฟ DOSSEG

- การกำหนดโมเดลของโปรแกรม (.MODEL) ไดเรคทีฟนี้จะทำให้แอสเซมเบลอร์จะจัดเซกเมนต์ให้เองตามชื่อโมเดลที่กำหนด โดยที่รูปแบบการใช้เป็นดังนี้

.MODEL ชื่อโมเดล

ชื่อโมเดลมี 5 โมเดล คือ small, medium, compact, large และ huge ซึ่งในการเขียนโปรแกรมแอสเซมบลีนี้ดังที่กล่าวแล้ว การกำหนดชื่อโมเดลจะต้องตรงกับภาษาซี จะเห็นว่าสำหรับเทอร์โบซีซึ่งมี Tiny โมเดลก็จะต้องใช้วิธีแรก คืออาศัย Template ในการเขียนโปรแกรม

- การกำหนดเซกเมนต์ ซึ่งได้แก่ .CODE, .DATA, .DATA? , .FARDATA, .FARDATA?, .CONST และ .STACK การใช้ไดเรคทีฟดังกล่าวนี้จะเป็นการบอกจุดเริ่มของเซกเมนต์ มีรูปแบบการใช้ดังรูปที่ 6

ไดเรคทีฟที่ต้องใช้ก็มี .CODE, .DATA, .DATA?, .FARDATA, .FARDATA? และ .CONST การใช้ไดเรคทีฟประกาศ

```
;**************** ASM12.asm *************
        dosseg
        .model small

        .data
;------------ Data in here -----------
string          db  'Now in assambly
                                    ',10,'$'
        .code
;-----------Program in here  ----------
        PUBLIC  _call_asm

_call_asm       proc    near
                mov     dx,offset string
                mov     ah,09h
                int     21h
                ret
_call_asm       endp

END
```

รูปที่ 7 ASM12.ASM โปรแกรมแอสเซมบลีตัวอย่างที่ 1 ใช้ไดเรคทีฟของแอสเซมเบลอร์ช่วยเขียน

รูปที่ 6

.STACK [size]	Stack segment
.CODE [name]	Code segment
.DATA	Initialized near-data segment
.DATA?	Unintialized near-data segment
.FARDATA [name]	Initialized far-data segment
.FARDATA? [name]	Uninitialized far-data segment
.CONST	Constant-data segment

เซกเมนต์นั้นสามารถใช้เรียงต่อไปเลย การใช้ไดเรคทีฟตัวใหม่จะถือเป็นการปิดเซกเมนต์ของไดเรคทีฟอันก่อนหน้า ส่วนไดเรคทีฟสุดท้ายจะถูกปิดด้วย END ไดเรคทีฟซึ่งอยู่สุดท้ายของไฟล์

ส่วนของโปรแกรมหรือ code นั้นจะอยู่หลังไดเรคทีฟ .CODE ตามปกติชื่อเซกเมนต์ของ code จะถูกกำหนดให้เป็นชื่อของไฟล์หรือโมดูล (ในกรณีโมเดล Medium, Large และ huge) ถ้าจะใช้หลาย ๆ เซกเมนต์จะต้องบอกชื่อของเซกเมนต์นั้นตามหลัง .CODE ด้วย

.DATA หรือ .FARDATA ใช้สำหรับประกาศเซกเมนต์สำหรับข้อมูลหรือตัวแปรที่ได้รับการกำหนดค่าก่อน ส่วน .DATA? หรือ .FARDATA? เป็นไดเรคทีฟสำหรับข้อมูลหรือตัวแปรที่ได้กำหนดค่า (Uninitialize) และ .CONST นั้นสำหรับข้อมูลหรือตัวแปรที่มีค่าคงที่ไม่เปลี่ยนแปลง

โปรแกรมตัวอย่างที่แล้วซึ่งเขียนโดยใช้ Template นั้นถูกเขียนโดยลักษณะที่ 2 ได้เป็นโปรแกรม ASM12.ASM ดังรูปที่ 7 ซึ่งนำไปใช้ได้กับโปรแกรมภาษาซี CCC1.C

การเขียนโปรแกรมแอสเซมบลีในลักษณะที่ 2 นี้จะพบว่าสะดวกกว่าในแบบ Template แต่ต้องใช้กับมาโครแอสเซมเบลอร์เวอร์ชัน 5.0 หรือใช้กับเทอร์โบแอสเซมเบลอร์เท่านั้น และสามารถนำออปเจ็คไฟล์ไปลิงค์ได้กับทั้งเทอร์โบซีและไมโครซอฟต์ซี

มีข้อสังเกตอยู่ตรงที่ชื่อฟังก์ชันในแอสเซมบลีจะมีเครื่องหมายขีดเส้นใต้นำหน้าชื่ออยู่ และลักษณะนี้จะถูกใช้กับชื่อฟังก์ชันและชื่อตัวแปร ที่ใช้ร่วมหรือให้ภาษาซีใช้ เป็นเพราะในตัวคอมพายเลอร์เป็นตัวจัดการเพื่อป้องกันการสับสนระหว่างชื่อภายในของคอมพายเลอร์ ตัวอย่างโปรแกรม CCC2.C และ ASM2.ASM ในรูปที่ 8 และ 9 ในแอสเซมบลีจะเป็นการใช้ตัวแปรร่วมกับของภาษาซีที่กำหนดในโปรแกรมภาษาซี แทนการทำงานของโปรแกรมจะเหมือนเดิม

การส่งผ่านพารามิเตอร์

การส่งผ่านพารามิเตอร์ในภาษาซีนั้นจะส่งผ่านทางสแต็ก การส่งผ่านจะ push พารามิเตอร์ลงสแต็กจากขวาไปซ้ายตามลำดับ (จะไม่กล่าวถึงการส่งผ่านพารามิเตอร์แบบปาสคาล ในการกำหนดด้วย pascal) ตัวอย่างเช่น

```
func (a,b,c);
```

c จะถูก push ลงสแต็กก่อนแล้วตามด้วย b และ a ขนาดของข้อมูลที่ต้อง push ลงสแต็กขึ้นกับชนิดของข้อมูล ซึ่งได้สรุปให้ในตารางที่ 3

และการอ้างถึงข้อมูลที่ส่งมาทางสแต็กมักจะใช้รีจิสเตอร์ BP เป็นตัวชี้ โดยทำการเก็บค่าใน BP ลงสแต็กก่อน แล้วจึงนำค่าใน SP มาใส่ BP แทน การใช้งานรีจิสเตอร์ตัวอื่นทำได้ตามปกติ แต่ว่ามีอยู่รีจิสเตอร์อยู่ 2 ตัว คือ SI และ DI

ถ้ามีการใช้งานจะต้องเก็บลงสแต็กและก่อนออกจากฟังก์ชันจะต้องคืนค่าเดิม เช่นเดียวกับ BP

การส่งค่ากลับของฟังก์ชัน

การส่งค่ากลับของฟังก์ชันในภาษาส่วนใหญ่จะส่งค่าผ่านทางรีจิสเตอร์ ตามชนิดของค่าที่ส่งจากฟังก์ชัน โดยสรุปได้ดัง

```
#include        <stdio.h>

extern  void    f(void);

char    string[]="Text in CCC2.C\r\n$
main()
{
        puts("Call F()");
        f();
        puts("Exit F()");
}
```

รูปที่ 8 CCC2.C โปรแกรมภาษาซี ตัวอย่างที่ 2

```
        dosseg
        .model  small

        .data
extrn   _string:byte
        .code
        public  _f
_f      proc    near
        push    bp
        mov     bp,sp
        mov     dx,offset _string
        mov     ah,9
        int     21h
        mov     sp,bp
        pop     bp
        ret
_f      endp
        end
```

รูปที่ 9 โปรแกรมแอสเซมบลี ตัวอย่างที่ 2

ตารางที่ 3 จำนวนไบต์ของชนิดตัวแปรหรือข้อมูลที่ต้องใส่ลงในสแต็ก

ชนิด	จำนวนไบต์
char	2
short	2
signed char	2
signed short	2
unsigned char	2
unsigned short	2
int	2
signed int	2
unsigned int	2
long	4
unsigned long	4
float	4
double	4
(near) pointer	8 (offset only)
(far) pointer	4 (segment and offset)

ตารางที่ 4

เพื่อความเข้าใจในเรื่องของการส่งผ่านพารามิเตอร์ การอ้างถึงพารามิเตอร์ และการส่งค่ากลับของฟังก์ชัน เราจะศึกษาดูถึงตัวอย่างโปรแกรมภาษาซี โดยจะคอมพายล์ให้ได้เป็นแอสเซมบลี (ใส่ option-S ในเทอร์โบซี และ option-Fa ในไมโครซอฟต์ซี) ดังในรูปที่ 10 ซึ่งจะแสดงถึงค่าในสแต็กขณะที่มีการเรียกฟังก์ชันด้วย

สำหรับ Return address นั้นขนาดมันขึ้นอยู่ฟังก์ชัน ถ้าฟังก์ชันเป็น near, Return address จะมีขนาด 2 ไบต์ แต่ถ้าเป็น Far จะมีขนาดเป็น 4 ไบต์ เพื่อความเข้าใจขอเพิ่มโปรแกรมตัวอย่างอีกสักโปรแกรม ในรูปที่ 11 โปรแกรม CCC3.C และในรูปที่ 12 โปรแกรม ASM3.ASM โดยซีจะเรียกโปรแกรมคูณซึ่งเป็นแอสเซมบลีโดยส่งค่าไปให้

ถึงตอนนี้แล้วก็คิดว่าท่านผู้อ่านคงสามารถเขียนโปรแกรมแอสเซมบลีเชื่อมโยงกับโปรแกรมภาษาซีได้แล้ว คงจะทิ้งตัวอย่างสุดท้ายไว้ให้ศึกษากันในรูปที่ 13 โปรแกรม CCC4.C และในรูปที่ 14 โปรแกรม ASM4.ASM โปรแกรมสุดท้ายจะเป็นการกำหนดวันและเวลาจากดอสทางแอสเซมบลีให้ภาษาซี โดยทางโปรแกรมภาษาซีจะส่งแอดเดรสของโครงสร้างข้อมูลไปให้แอสเซมบลี ลองสังเกตการอ้างถึงข้อมูลนะครับ รายละเอียดต่าง ๆ ถ้าท่านผู้อ่านยังไม่เข้าใจ อ่านจากหนังสือหรือเอกสารอ้างอิงที่ให้ไว้ได้ครับ

เอกสารอ้างอิง

1. HERBERT SCHILDT, "Advanced TURBO C", Osborne Mc Graw-Hill, Inc. 1987
2. "Microsoft Marco Assembler 5.0", Microsoft Corporation, 1987
3. "Microsoft C Compiler 5.0", Microsoft Corporation, 1987
4. "TURBO C Compiler 2.0", Borland International, 1988
5. "TURBO ASSEMBLER 1.0", Borland International, 1988

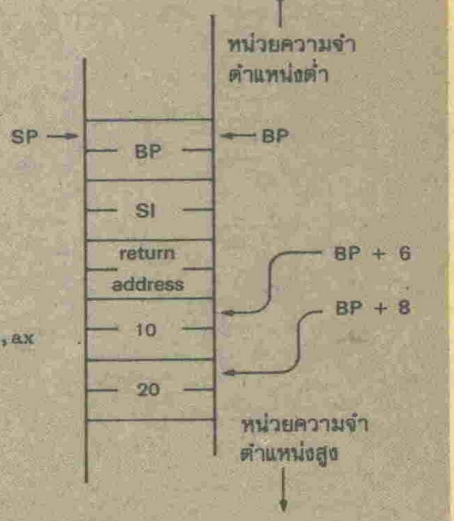
ตารางที่ 4 การส่งค่ากลับของฟังก์ชันในภาษาซี

ชนิด	รีจิสเตอร์และความหมาย
char	AX
unsigned char	AX
short	AX
unsigned short	AX
int	AX
unsigned int	AX
long	low-order word in AX High-order word in DX
unsigned long	Low-order word in AX High-order word in DX
float	Low-order word in AX High-order word in DX
double	Return on 8087 stack or top of stack in emulator
struct & union	Address to value
(near) pointer	AX
(far) pointer	Offset in AX, Segment in DX

รูปที่ 10 ตัวอย่างโปรแกรมภาษาซี ซึ่งถูกคอมพายล์เป็นภาษาแอสเซมบลี และค่าในสแต็กขณะการเรียกใช้ฟังก์ชัน

```
int sum;

main ()
{
     sum = add (10,20);
}

int add (int a,int b)
{
     int t;

     t = a + b;
     return (t);
}
```

```
_main     proc     near
          mov      ax,20
          push     ax
          mov      ax,10
          push     ax
          call     near ptr _add
          pop      cx
          pop      cx
          mov      word ptr dgroup:_sum,ax
          ret
_main     endp

_add      proc     near
          push     si
          push     bp
          mov      bp,sp
          mov      si,word ptr [bp+6]
          add      si,word ptr [bp+8]
          mov      ax,si
          pop      bp
          pop      si
          ret
_add      endp
```



รูปที่ 11 CCC3.C โปรแกรมภาษาซีตัวอย่างที่ 3

```
/****** Second Example CCC3.C ******/

#include        <stdio.h>

extern  long mul(int,int);

main()
{
        puts("Call mul(10,20)");
        printf("3000x3000=%ld\n",mul(3000,3000));
        puts("Exit mul()");
}
```

```
        dosseg
        .model  small
        .code
        public  _mul
arg1    equ     [bp+4]
_mul    proc    near
        push    bp
        mov     bp,sp
        mov     ax,arg1
        mov     dx,arg1+2
        mul     dx
        mov     sp,bp
        pop     bp
        ret
_mul    endp
        end
```

รูปที่ 12
ASM3.ASM
โปรแกรมแอสเซมบลี
ตัวอย่างที่ 3

รูปที่ 13 CCC4.C โปรแกรมภาษาซีตัวอย่างที่ 4

```
/****** Second Example CCC4.C ******/

#include        <stdio.h>

struct  date {
        int     year;   /* 1980 - 2099 */
        char    month;  /* 1 - 12 */
        char    day;    /* 1 - 31 */
        char    day_of_week; /* 0 = sunday */
}       rdate;

struct  time {
        char    hour;   /* 0 - 23 */
        char    minute; /* 0 - 59 */
        char    second; /* 0 - 59 */
        char    hundredth;      /* 0 - 99 */
}       rtime;

extern  void    gettime(struct time *);
extern  void    getdate(struct date *);

void showdate()
{
        static  char *day_name[]={
                "Sunday","Monday","Tuesday",
                "Wednesday","Thursday","Friday",
                "Saturday"};
        static  char *month_name[]={
                "January","February","March",
                "April","May","June","July",
                "August","September","October",
                "November","December"};
                printf("Today is %s %s %d,%d\n",
                day_name[rdate.day_of_week],
                month_name[rdate.month],
                rdate.day,rdate.year);

}
```

```
void showtime()
{
        printf("Current time is %02d:%02d:%02d.%02d\n"
                rtime.hour,rtime.minute,
                rtime.second,rtime.hundredth);

}

main()
{
        getdate(&rdate);
        showdate();
        gettime(&rtime);
        showtime();
}
```

รูปที่ 14 ASM4.ASM โปรแกรมภาษาแอสเซมบลีตัวอย่างที่ 4

```
;*************** ASM4.ASM *************

date    STRUC
        year    DW      ?
        month   DB      ?
        day     DB      ?
        day_of_week     DB      ?
date    ENDS

time    STRUC
        hour    DB      ?
        minute  DB      ?
        second  DB      ?
        hundredth       DB      ?
time    ENDS

        DOSSEG
        .MODEL  small
        .CODE
        PUBLIC  _getdate,_gettime
_getdate        PROC    NEAR
        push    bp
        mov     bp,sp
        push    si
        mov     ah,2ah          ; get date
        int     21h               function call number
        mov     si,[bp+4]       ; call DOS
        mov     [si].year,cx    ; get passed
        mov     [si].month,dh     parameter
        mov     [si].day,dl
        mov     [si].day_of_week,al
        pop     si
        mov     sp,bp
        pop     bp
        ret
_getdate        ENDP

_gettime        PROC    NEAR
        push    bp
        mov     bp,sp
        push    si      ; save before use
        mov     ah,2ch  ; get time
                          function call number
        int     21h     ; call DOS
        mov     si,[bp+4]
        mov     [si].hour,ch
        mov     [si].minute,cl
        mov     [si].second,dh
        mov     [si].hundredth,dl
        pop     si      ; restore
        mov     sp,bp
        pop     bp
        ret
_gettime        ENDP

        END
```